In the Drawings

A substitute FIG. 5 has been included

Remarks

Claims 1-21 are at issue. Claims 1-21 stand rejected under 35 USC 101 as being directed to non-statutory subject matter. Claims 8 & 9 stand rejected under 35 USC 112, second paragraph. Claims 1-21 stand rejected under 35 USC 102(e) as being anticipated by Ye et al (US Pat. Pub. 2004/0083242).

Drawings

A new FIG. 5 is included.

Specification

The Examiner's rejections are not based on the law or the MPEP. There are no 35 USC 112 issues for using a synonym or class with the same reference numeral. For instance, a switch 10 may also be referred to a transistor 10 or a motor 10 may be referred to a power source 10. These do not raise 112 issues. The Examiner's rejections have no basis in the law or logic and must be withdrawn.

As explained on page 6, lines 10-11 the transformer program is a transformer document. FIG. 5 is an example of a transformer document or program. The applicants disagree that any change is necessary.

Line 5 of page 6 of the specification states the "output may be a streamed XML 34 or an XML system." This is correct and consistent. The output may be streamed or it may be a file. This is perfectly clear to those skilled in the are and no change is necessary.

35 USC 101

"Statutory"

The present application is a "useful process." The present application is clearly a machine. Software is just a way of temporarily wiring an electric circuit (computer) to perform a specific task. Electrical circuits are machines. Corporations spend millions

of dollars trying to solve the problem of moving data from one system to another system – therefore this is clearly a useful invention.

"MPEP 706.03(a)"

MPEP section 706.03(a) states three categories of subject matter that are non-statutory: 1) Printed Matter, 2) Naturally Occurring Article, and 3) Scientific Principle. The present application relates to computer system operating to perform a useful data conversion tool. Clearly, the application and the claims are not 1) Printed Matter, 2) A Naturally Occurring Article, and 3) A Scientific Principle. Claims 1-21 are enabled by the specification and therefore must be directed to statutory matter. The Patent Office has failed to grasp the simple fact that a computer program run by a general purpose computer could just as easily be a hardwired circuit. There is no question that a hardwired circuit is patentable. Clearly the Patent Office is still wasting everyone's time and money by not acknowledging that computer implemented **Products** are clearly statutory. A computer running a program that does not function to provide just "printed matter", is clearly a **Machine** as defined by the statute 35 USC 101.

The rejection of claims 1-21 under 35 USC 101 must be withdrawn.

35 USC 102

Claim 1 requires an "input text file." According to Wikipedia (See attached) a text file is a computer file which contains only ordinary textual characters with essentially no formatting. This is consistent with the specification of the present application, see FIG. 3 which is an example of a text file. Ye et al clearly disclose a system for converting "formatted data" see FIG. 2 of Ye and associated text.

Claim 1 furthermore requires that the "XML document does not contain every element that was in the input text." Ye shows converting every input text into an XML format. Claim 1 is clearly allowable over the prior art.

Claim 1 as amended also requires the transformer program have a plurality of compound statements. The Examiner suggests that every computer program contains "executable statements." Perhaps the applicant did not choose the correct terminology.

According to Wilkipedia (see attached) a statement may be either a "simple statement" or a "compound statement." The manner in which the present application uses the term "executable statement" is consistent with the definition of a "compound statement." Ye does not disclose the use of a "compound statements." Claim 1 is allowable.

Claim 2 requires that the input file be a structured document. According to the specification, page 5, line 15, "an example of structured text is a comma delimited file." The applicant may admit that this is discussed in the background of Ye.

Claim 3 requires that the input file be a semi-structured document. According to the specification, page 5, line 16, "an example of semi-structured text is a windows initialization file used by computers." This is clearly not discussed in Ye. Claim 3 is allowable.

Claim 4 requires that the input file have at least two formats. According to the specification, page 5, line 16, "In one embodiment, the text file may contain multiple different formats. For instance, it might have a part that is comma delimited and another part that is delimited by square brackets." This is clearly not discussed in Ye. Claim 4 is allowable.

Claim 5 requires a field separator command. The Examiner points to a field separator, but does not show a field separator command. Claim 5 is clearly allowable.

Claims 6-8 are allowable as being dependent upon an allowable base claim.

Claim 9 requires the "text to XML commands" include a tree hierarchy. First of all Ye never discloses any "text to XML commands." The Examiner's suggestion that XML inherently has a tree structure while correct is irrelevant – this says nothing about the "text to XML commands". For instance the conversion program could be written in C commands that do not have a tree structure. Claim 9 is clearly allowable.

Claim 10 requires the input be a streaming text. The Examiner points to paragraph 184 of Ye, which never mentions streaming. Not all conversion systems are capable of handling streaming data and there is no suggestion in Ye that he can handle streaming data. Claim 10 is clearly allowable.

Claim 11 is allowable for the same reasons as claim 10.

Claim 12 requires a wizard to define the transformer program. The Examiner points to paragraph 77. However, this paragraph does not discuss a transformer program or a wizard to help setup the transformer program.

Claim 13 is allowable as being dependent upon an allowable base claim.

Claim 14 as amended also requires the transformer program have a plurality of compound statements. The Examiner suggests that every computer program contains "executable statements." Perhaps the applicant did not choose the correct terminology. According to Wilkipedia (see attached) a statement may be either a "simple statement" or a "compound statement." The manner in which the present application uses the term "executable statement" is consistent with the definition of a "compound statement." Ye does not disclose the use of a "compound statements." Claim 14 is allowable.

In addition, claim 14 requires that there be a match command that matches a "regular expression." No such match command is discussed in Ye.

Claim 15 requires a wizard. The Examiner points to paragraph 77. However, this paragraph does not discuss a wizard. Claim 15 is allowable.

Claim 16 requires selecting a field separator command. Ye never discusses "selecting" a field separator command. Claim 16 is allowable.

Claim 17 is allowable as being dependent upon an allowable base claim.

Claim 18 requires that the input file have at least two formats. According to the specification, page 5, line 16, "In one embodiment, the text file may contain multiple different formats. For instance, it might have a part that is comma delimited and another part that is delimited by square brackets." This is clearly not discussed in Ye. Claim 18 is allowable.

Claim 19 as amended also requires the transformer program have a plurality of compound statements. The Examiner suggests that every computer program contains "executable statements." Perhaps the applicant did not choose the correct terminology. According to Wilkipedia (see attached) a statement may be either a "simple statement" or a "compound statement." The manner in which the present application uses the term "executable statement" is consistent with the definition of a "compound statement." Ye does not disclose the use of a "compound statements." Claim 19 is allowable.

Claim 19 also requires a wizard for creating a transformer document. Ye never discusses a transformer document or a wizard for creating a transformer document.

Claim 20 requires a section command. Ye never discusses a section command and the Examiner has not attempted to point to any part of Ye to show this. Claim 20 is allowable.

Claim 21 is allowable as being dependent upon an allowable base claim.

Ye

While Ye is concerned with data conversion, he uses data location to find the data (See paragraph 96). This is inherently not possible with streaming data where the location cannot be predicted. Because the location feature is used to define where the data is located (see paragraph 108) Ye is only applicable to data presented in a display format, which is not a text file. The present application is concerned with data in a text file or streamed text file, not in a display format. Ye would not be capable of solving the problem addressed by the present application, since the data to be converted is never in a "location" or presentation format. These differences are clearly pointed out in the claims.

Prompt reconsideration and allowance are respectfully requested.

Respectfully submitted,

(Snyder)

By _____
Attorney for the Applicant
Dale B. Halling
Registration No.: 38,170
Customer No.: 25,007
Phone: (719) 447-1990
Fax: (719) 447-9815

I hereby certify that an Response is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner of Patents and Trademarks, P.O. Box 1450 Alexandria, VA 22313-1450, on:

9/29/06
Date

_____
Signature (Dale Halling)

Answers.com™

statement

Wikipedia

statement (programming)

In computer programming a **statement** can be thought of as the smallest standalone element of an imperative programming language. A program is formed by a sequence of one or more statements. A statement will have internal components (eg, expressions).

Many languages (eg, C) make a distinction between statements and definitions, with a statement only containing executable code and a definition declaring an identifier. A distinction can also be made between simple and compound statements; the latter may contain statements as components.

The following are the major generic kinds of statements with examples in typical imperative languages:

definitions and declarations

- **definition:** `TYPE SALARY = INTEGER`
- **declaration:** `VAR A:INTEGER`

simple statements

- **assignment:** `A := A + 1`
- **call:** `CLEARSCREEN()`
- **return:** `return 5;`
- **goto:** `goto 1`
- **assertion:** `assert(ptr != NULL);`

compound statements

- **statement block:** `begin WRITE('Number? '); READLN(NUMBER); end`
- **if-statement:** `if A > 3 then WRITELN(A) else WRITELN("NOT YET") end`
- **switch-statement:** `switch (c) { case 'a': alert(); break; case 'q': quit(); break; }`
- **while-loop:** `while NOT EOF DO begin READLN end`
- **do-loop:** `do { computation(&i); } while (i < 10);`
- **for-loop:** `for A:=1 to 10 do WRITELN(A) end`

In most languages statements contrast with expressions in that the former do not return results and are executed solely for their side effects, while the latter always return a result and often do not have side effects at all. Among imperative programming languages, Algol 68 is one of the few in which a statement can return a result. In languages which mix imperative and functional styles, such as the Lisp family, the distinction between expressions and statements is not made: even expressions executed in sequential contexts solely for their side effects and whose return values are not used are considered 'expressions'.

The syntax and semantics of statements is usually specified in the definition of the programming language and cannot be changed in a program. Only a few programming languages allow user defined statements or overloading of statements.

Typical programming languages with statements are Ada, ALGOL, BASIC, C, [[C++]], COBOL, Eiffel, Fortran, Java, JavaScript, Modula, Oberon, Pascal, Perl, PL/I, Python, REXX, Ruby, Seed7, Simula and Tcl.

Answers.com™

expression

Wikipedia

expression (programming)

An **expression** in a programming language is a combination of values, variables, operators, and functions that are interpreted (*evaluated*) according to the particular rules of precedence and of association for a particular programming language, which computes and then produces (*returns*, in a stateful environment) another value. The expression is said to *evaluate to* that value. As in mathematics, the expression *is* (or can be said to *have*) its evaluated value; the expression is a representation of that value.

Expressions may or may not have side-effects and lose the referential transparency with them.

See also

- Statement (programming) (contrast)
- Boolean expression
- Expression (mathematics)
- evaluation strategy

References

- Expression in The Free On-line Dictionary of Computing, Editor Denis Howe.

This entry is from Wikipedia, the leading user-contributed encyclopedia. It may not have been reviewed by professional editors (see full disclaimer)

Donate to Wikimedia

Mentioned In

expression is mentioned in these AnswerPages:

Expression (Shopping)	repression
quintic (mathematics)	doxology
acclamation	applause
laugh, laughter	Epson Expression Professional 1680 (Shopping)
espressivo	expressionless
	More>

44

```
46 <?xml version="1.0"?>
48 <tx:transform xmlns:tx="http://www.xaware.com/2003/01/text-to-xml"
   version="1.0">
       <!-- initialize and set the document element -->
50     <tx:template match="BEGIN">
51       <tx:variable name="FS" value="',"/>
         <addresses>
           <tx:continue/> 52
         </addresses>
       </tx:template>
       <!-- skip the first record because it is just a header -->
54     <tx:template match="NR = 1">
56       <tx:next/>
         </tx:template>
         <!-- add address for category customers -->
58       <tx:template match="F[7]  = 'customers'">
62         <address name=" {F[2]}">
64           <name>
66             <tx:value-of expr="F[1]"/><tx:text> </tx:text>
68             <tx:value-of expr="F[2]"/>
70           </name>
72           <street><tx:value-of expr="F[3]"/></street>
             <city><tx:value-of expr="F[4]"/></city>
             <state><tx:value-of expr="F[5]"></state>
             <xip><tx:value-of expr="F[6]"/></zip>
           </address>
         </tx:template>
   </tx:transform>
```

FIG. 5

80

```
[App Settings]
window=maximized
advanced mode=true
install=C:\someapp

[Tool Settings]
tool1 driver =  lib\somedll.dll  /s=46  /t="fast"
```

FIG. 6